HARDWARE MODELLING AND SIMULATION USING AN OBJECT-ORIENTED METHOD

Frédéric MALLET*, Fernand BOERI* Senior Member IEEE, Jean-François DUBOC**
* Laboratoire I3S, UPRES_A 6070 CNRS, Université de Nice-Sophia Antipolis, 41 Bd Napoléon III
06041 Nice Cédex France. Tel : (+33) 4 93 21 79 58 e-mail : fmallet@i3s.unice.fr, boeri@unice.fr
** VLSI Technology inc., 505 Route des Lucioles, Sophia-Antipolis 06560 Valbonne.
Tel : (+33) 4 92 96 11 81 e-mail : jean-francois.duboc@sophia.europe.vlsi.com

KEYWORDS

Simulation, Modelling, Object-oriented, Hardware architectures, Performance evaluation.

ABSTRACT

In order to reduce the cost, the time-to-market and to make the most pertinent choices, it becomes essential to allow designers to evaluate, very soon in the design phase, a given application performances with respect to the targetted architecture. So, we have decided to build a modelling and simulation environnement in order to evaluate digital hardware architecture performances. We considered the requisite number of cycles for processing a given application with a simple model of the architecture. In this project, we need to increase the reusability with an adjustable abstraction level. So, we decided to use object-orientation concepts to build our environnement. Then, reusing already designed components, designers will be able to build models with a level of abstraction which fit theirs goals. So, our main objective and the greatest part of out work was to define a generic object-oriented model of digital architectures. This paper mainly consists in the explanation of this model which is designed to help us to implement a visual modelling and simulation environnement.

INTRODUCTION

Digital hardware architectures and embedded applications are becoming more and more complex. The very high integration rate and the increasing complexity lead designers to search for new tools and methods. Performances of a given application and architecture have to be evaluated sooner and sooner in the design phase. So we have decided to build a modelling and simulation environnement in order to evaluate digital hardware architecture performances. We aim mainly at evaluating the requisite number of cycles for processing a given application with a simple model of the architecture.

Increase the reusability and adjust the abstraction level of models is not a recent problem and object-oriented methods were born to solve this problem for software systems [Swa95]. For a few years, lots of international projects (RASSP [ETO96] from the American Department of Defense; POLIS [BER96] from the University of Berkeley, California) have tried to find an object-oriented solution for hardware systems. One result of these projects consists in the definition of an object-oriented layout upon VHDL (OO-VHDL) [BaB96] [AMS95] [BeD97] which is with Verilog one of the two most used hardware description languages. But, VHDL is a very heavy language designed to synthesize hardware systems and it includes lots of mechanisms which are useless in first steps of the design phase. So, we have chosen to construct upon Java - an entire and pure object-oriented language - mechanisms which will permit designers to model and simulate hardware architectures. Object-orientation includes abstraction and encapsulation mechanisms and allows polymorphism and inheritance. Then, reusing already designed components, designers will be able to build models with a level of abstraction which fits theirs goals. The abstraction level will be choosen by the designer during the modelling phase by an incremental way and depending on the expected results.

Now, we expose our method applied to a simple example from the simulation context to the lowest description level which is here the gate level. Using this example we introduce every words we need to construct our model. And then we present our deductions and the resulting generic object-oriented model.

OUR METHOD

We have proceeded in several steps. The first step was to define the main characteristics of the targetted digital architectures. We would have liked to have a generic description with a few constraints in order to be able to study not only simple processors but also multicore or distributed architectures. The second step was to determine basic components of these targetted architectures in order to make a standard basic library of reusable components. The third step was to choose and apply an object-oriented design method in order to construct a generic object-oriented framework for the description and the simulation of the targetted architectures and applications. This step constitutes the greatest part of our work. The last step is going to be the construction of a visual modelling and simulation interface which can use the proposed object-oriented model. This interface will be described entirely in a following paper with an application to a real Digital Signal Processor (DSP).

SIMULATION CONTEXT

By example, we would have liked to simulate a DSP core. Then we need some external components (reset, clock, ROM) which constitute the simulation context for the core (cf. Figure 1).

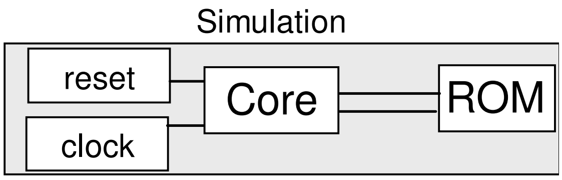

Figure 1 - Simulation context

This context model was implemented with a Java application named Simulation. In this class, each component (reset, clock, Core, ROM, Signal) is represented by another Java class. We just need to instantiate these classes and to make links between components as shown by the following source code.

```
class Simulation extends Frame {
public static void main(String arg[]) {
    Core pine = new Core();
    ROM mem = new ROM();
    Actuator rst = new Actuator("reset");
    Actuator clk = new Actuator("clock");
    Signal reset = new Signal();
    reset.linkToPort(rst.getPort("out"));
    reset.linkToPort(pine.getPort("reset"));
    Signal clock = new Signal();
    clock.linkToPort(clk.getPort("out"));
    clock.linkToPort(pine.getPort("clk"));
    Signal ppan = new Signal();
    ppan. linkToPort(pine.getPort("ppan"));
    ppan. linkToPort(mem.getPort("addr"));
    Signal idp = new Signal();
    idp. linkToPort(pine.getPort("idp"));
    idp. linkToPort(mem.getPort("out"));
}}
```

BASIC CLASSES FOR THE MODEL

It appears that we need to define several classes in order to represent different parts of the modelled system. So we consider digital hardware architectures as **material components**. The description of these components can be a behavioural description or a structural description. In the example above, we distinguish three material components, one with a structural description (Core) and two with a behavioural description (Actuator, ROM). The communication with other components is done across some **port**s. These ports are connected by **signal**s which transport different **value**s. These values can be specialised into several sub-classes (assembly instructions, integers, events, bit values, hardware levels).
In the example, the ROM component can easily be represented with a behavioural description language. In the other hand, the Core component should be represented with a structural description.
Components which description is behavioural are called elementary material components and constitute the leafs of our hierarchy. Other components are represented with the **materialContainer** class.

THE MATERIAL CONTAINER

According to this, in what follows we give a rough sketch of the current Java implementation for the Core component which is a material container.

```
class Core extends materialContainer {
public Core() {
    /* variables declaration ... */
    /* components instantiation */
    addMaterialComponent(new PCU());
    addMaterialComponent(new CU());
    addMaterialComponent(new DAU());
    /* signal connections */
    addSignal(GDP = new Signal());
    GDP. linkToPort(pcu.getPort("gdp"));
    GDP. linkToPort(cu.getPort("gdp"));
    GDP. linkToPort(dau.getPort("gdp"));
    /* others .... */
}}
```

A material container only serves to encaspulate material components and signals; it allows a structural description for material components. These components could either be elementary material components or material containers. So the materialContainer class has to inherit from the materialComponent class.

ELEMENTARY COMPONENTS

We said before there are components with a simple behaviour like the ROM component. These components were to constitute entities the description of which has the lowest abstraction level. The ROM code is not very interesting to illustrate our behavioural model so the following Java code presents the description of an edge-triggered register (cf. Figure 2) with an asynchronous reset.

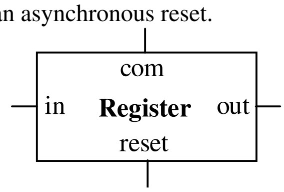

Figure 2 - Edge-triggered Register

```
class Register extends EdgeComponent {
    public Register () {
        addRisingEdgePort("com");
        addFallingEdgePort("reset");
        addPort("in"); addPort("out");
    }
    void report(Value v, String sender) {
        if (sender.equals("reset"))
            value = new IntValue(0);
        else if (sender.equals("com"))
            value = getPort("in").read();
        emit("out", value);
}}
```

The EdgeComponent class inherits from the materialComponent class. It describes material components the behaviour of which is commanded when an edge event occurs on one of the ports. Each port can be responsible for the component sensitivity. In the case of the Register component, the builder declares ‘in’ and ‘out’ ports as unsensitive ports, ‘com’ port as sensitive on a rising edge event and ‘reset’ port as sensitive on a falling edge event. So, when an event occurs on a sensitive port, the report method of the component is executed.

USING A LIBRARY

The preceding example shows how we can use existing components and how we can create new components. After the study of several digital architecture examples, it appears some components which are often used. These components have been implemented and added to a library of standard components (multiplexer, ALU, register, multiplier, memory). Moreover, in order to construct the model of a DSP core more components were needed (decoder, PC, DAU, PCU, CU). So we give some terminal components which can be reused because they are objects inheriting from the materialComponent class. But for each new studied system, the user will have to construct its own elementary component library.

THE GENERIC OBJECT-ORIENTED MODEL

Until now we shown how we can deduce some digital hardware architecture properties. In a more general way, we have studied several digital architectures and we deduced some general properties. These properties have been modelled using the Object Modelling Technique (represented in Figure 3). Pine and Oak are digital signal processors and ARM is a RISC processor.

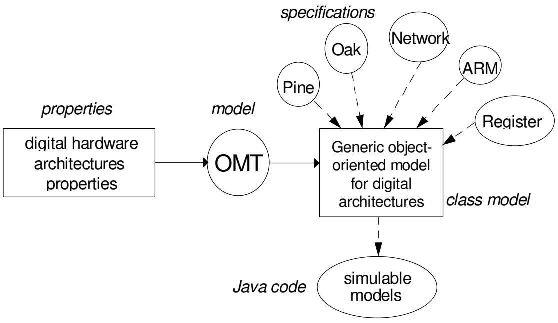

Figure 3 - Representing genericity with OMT

Following this method, we construct a generic object-oriented model for digital hardware architectures. This model is presented by the Figure 4 and contains every deduced properties.

Using this unique model we are able to construct simulable models from specifications of each specific architecture. The application of this generic model to a specific architecture can be boring so we have built a graphical interface which is able to apply it in a semi-automatic way.

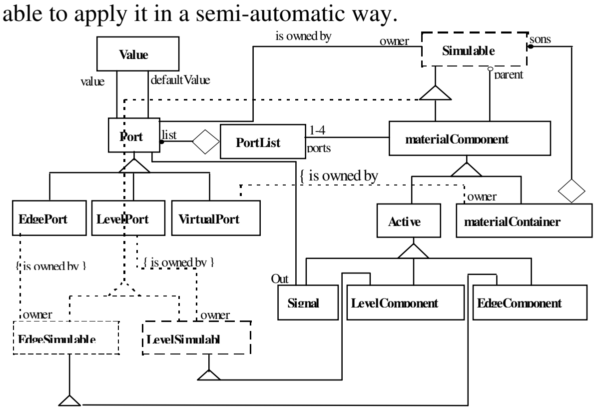

Figure 4 - Generic object-oriented model

CONCLUSION AND FUTURE WORKS

With this generic object-oriented model we can describe each of the targetted digital hardware architectures from a basic specification like a simplified block diagram. Applying this technique, we had constructed in a few weeks a simple simulable model of a real digital signal processor core. This model uses a reduced subset of the real processor assembly language for the proposed object-oriented model. Results obtained for processing some classical DSP applications (FFT, Viterbi, FIR) with our model and with the real synthesized component are exactly the same. Then we are constructing a visual interface which can use the proposed object-oriented model. This is about to be done using the Java Development Kit 1.1.5 (or 1.2 if released) and the interface features will be described in a following paper. The Figure 5 illustrates this interface. Moreover, the Java language integrates lots of useful mechanisms to perform distributed communication. Then, we are going to apply the model to a real distributed application (a Vehicle Area Network) and to a multicore architecture. This could permit us to extend and improve our model.

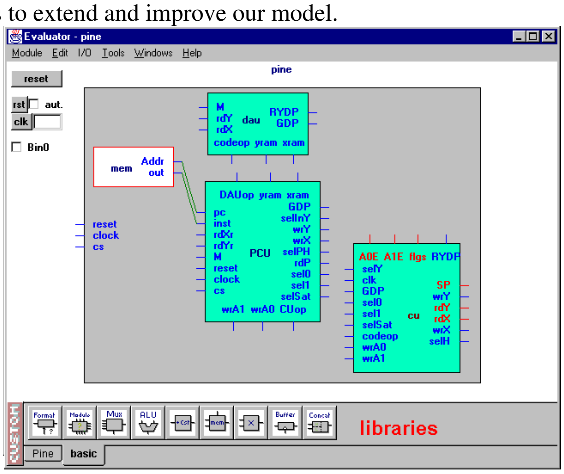

Figure 5 - graphical interface

REFERENCE

[AMS95] *Object-Oriented High-Level Modeling of System Components for the Generation of VHDL Code.*
Karlheinz Agsteiner, Dieter Monjau, Sören Schulze. 0-8186-7156-4/95. IEEE, 1995.

[BaB96] *A proposed Design Objectives Document for Object-Oriented VHDL.*
David L.Barton, Jean Michel Berge. The RASSP Digest - Vol. 3, September 1996.
http://rassp.scra.org/newsletters/96sep/news_18.html.

[BeD97] *Object Oriented Extensions to CHDL, The LaMI proposal.*
J.Benzakki, B.Djaffri.IFIP 1997. Chapman & Hall.p.334-347.

[Ber96] *A Framework for Harware-Software Co-Design of Embedded Systems : POLIS.*
http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html.

[ETO96] *The Rapid prototyping of Application Specific Signal Processors (RASSP) Program.*
http://eti.sysplan.com/ETO/RASSP.

[Swa95] *Object-Oriented VHDL Provides New Modeling and Reuse Techniques for RASSP.*
Dr. Sowmitri Swamy, Vista RASSP Program Manager. The RASSP Digest - Vol. 2, No. 1, 1st. Qtr. 1995
http://rassp.scra.org/newsletters/95q1/news_6.html